|  |
| --- |
| Experiment No. 5 |
| Exploring Files and directories: Python program to append data to existing file and then display the entire file |
| Date of Performance: 12/02/2024 |
| Date of Submission: 26/02/2024 |

**Experiment No. 5**

**Title:** Exploring Files and directories: Python program to append data to existing file and then display the entire file

**Aim:** To Exploring Files and directories: Python program to append data to existing file and then display the entire file

**Objective:** To Exploring Files and directories

**Theory:**

Directory also sometimes known as a folder are unit organizational structure in computer’s file system for storing and locating files or more folders. Python now supports a number of APIs to list the directory contents. For instance, we can use the Path.iterdir, os.scandir, os.walk, Path.rglob, or os.listdir functions.

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with Reading and Writing files.

**Working of open() function**

We use open () function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use open() function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: open(filename, mode). There are three kinds of mode, that Python provides and how files can be opened:

“ r “, for reading.

“ w “, for writing.

“ a “, for appending.

“ r+ “, for both reading and writing

**CODE:**

**1] File\_Handling Program:**

#operations based on mode of the files

#write function

f=open('icecream.txt','w')

str = input('enter text: ')

f.write(str)

f.close()

#append function

f=open('icecream.txt','a')

str2 = input('enter text: ')

f.write(str2)

f.close()

#read function

f=open('icecream.txt','r')

strU = f.read()

print(strU)

f.close()

#seek functions according to the byte position

f=open('icecream.txt','r')

stra = f.seek(6)

print("The seeked string from 4th byte is ", f.read())

f.close()

#ReadLine

f = open('icecream.txt', 'r')

print("The first line is : ", f.readline())

f.close()

#------------operations based on strings-----------------

f=open('icecream.txt','r')

#print the lines in file

for line in f:

print(line)

#print words in line of file

for line in f:

word = line.split()

print(word)

f.close()

#count of words , lines and characters

f=open('icecream.txt','r')

cw = cl = cc = 0

for line in f:

word = line.split()

cl = cl+1

cw = cw+ len(word)

cc = cc+ len(line)

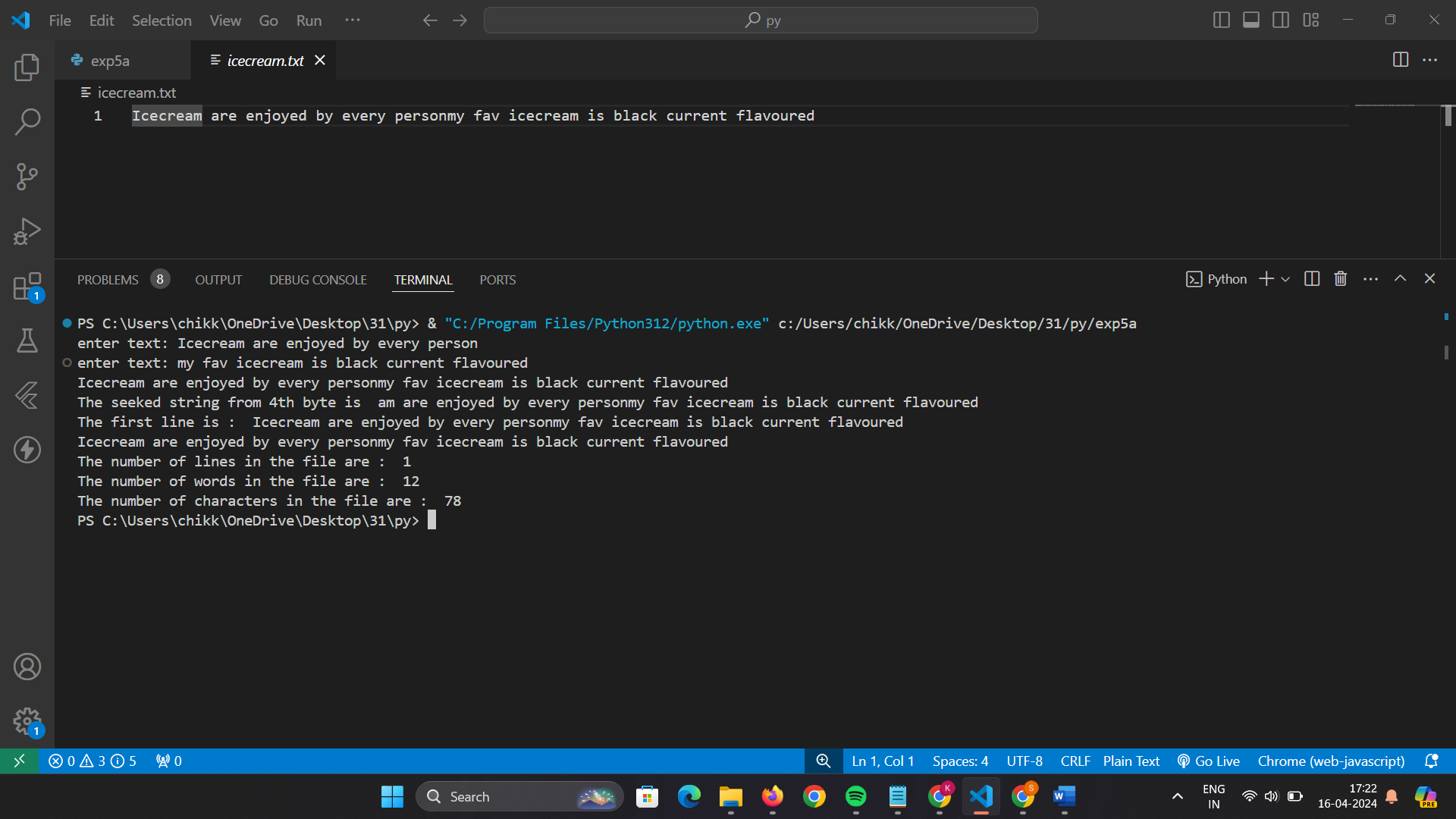
print("The number of lines in the file are : ", cl)

print("The number of words in the file are : ", cw)

print("The number of characters in the file are : ", cc)

f.close()

**File\_Handling Output:**

****

**2] Directories Program:**

import os

# Get current working directory

current\_directory = os.getcwd()

print("Current working directory:", current\_directory)

# Create a new directory

new\_directory = "test\_directory"

os.mkdir(new\_directory)

print("Created directory:", new\_directory)

# Create subdirectories

os.makedirs("test\_directory/subdir1/subdir2")

print("Created subdirectories")

# Change current working directory

os.chdir(new\_directory)

print("Changed working directory to:", os.getcwd())

# Remove a directory

os.rmdir("subdir1/subdir2")

print("Removed subdirectories")

# Remove multiple directories recursively

os.rmdir("subdir1")

print("Removed multiple directories")

# Rename a directory

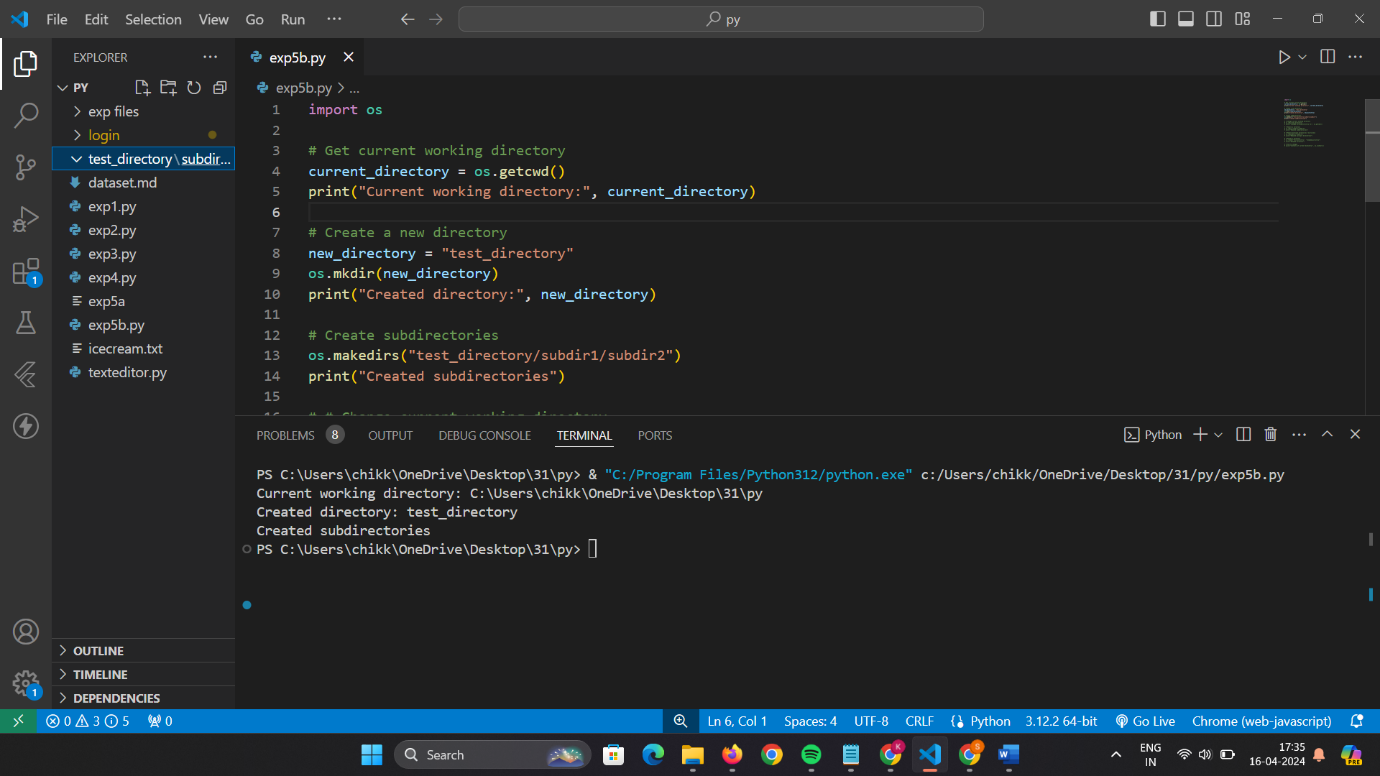
os.rename("test\_directory", "renamed")

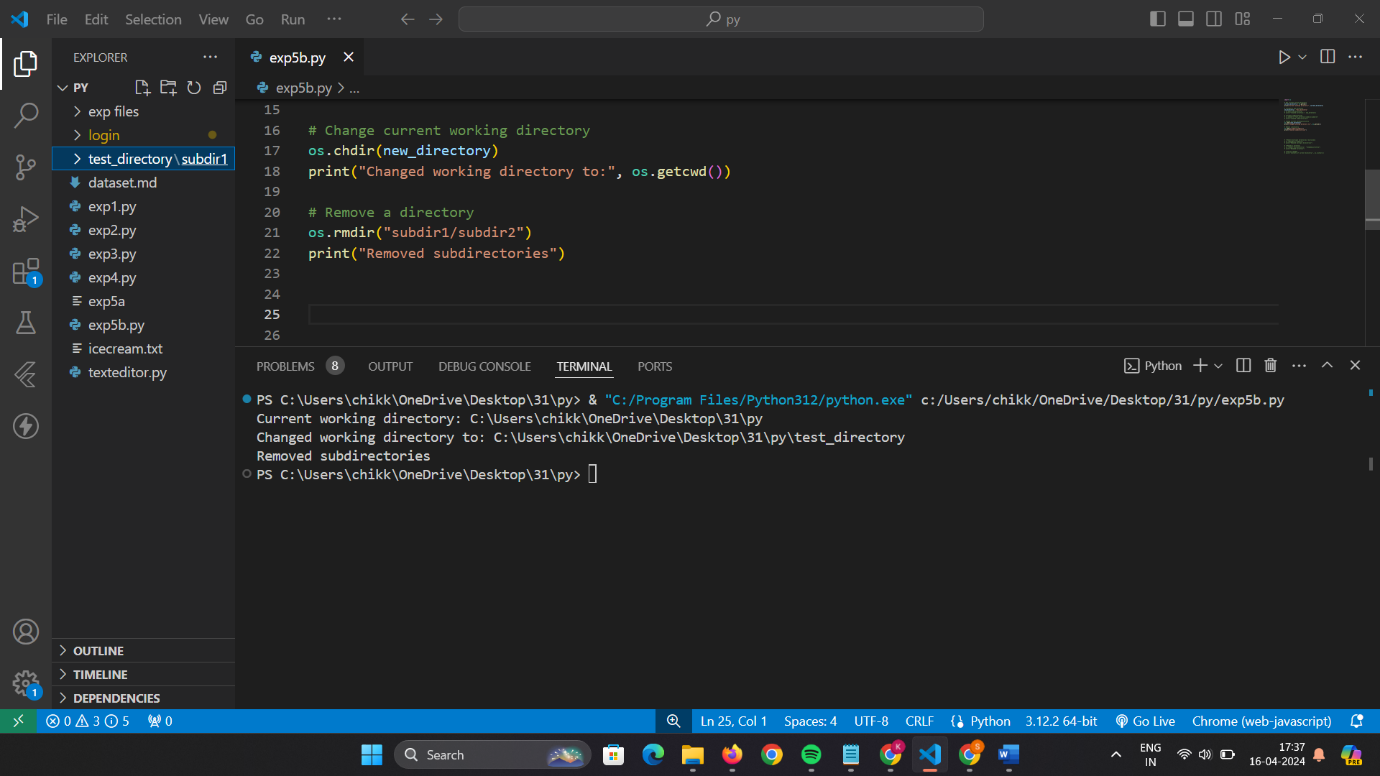
print("Renamed directory")

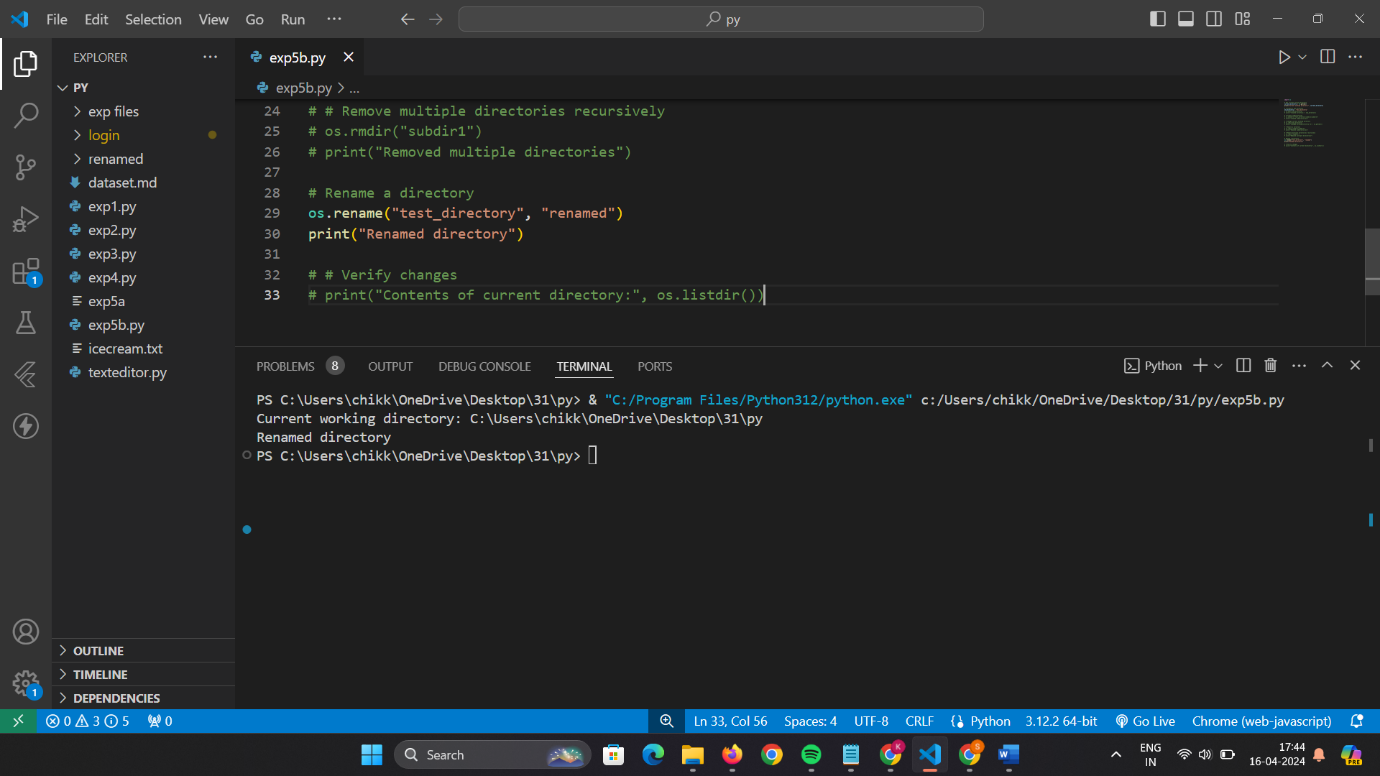
# Verify changes

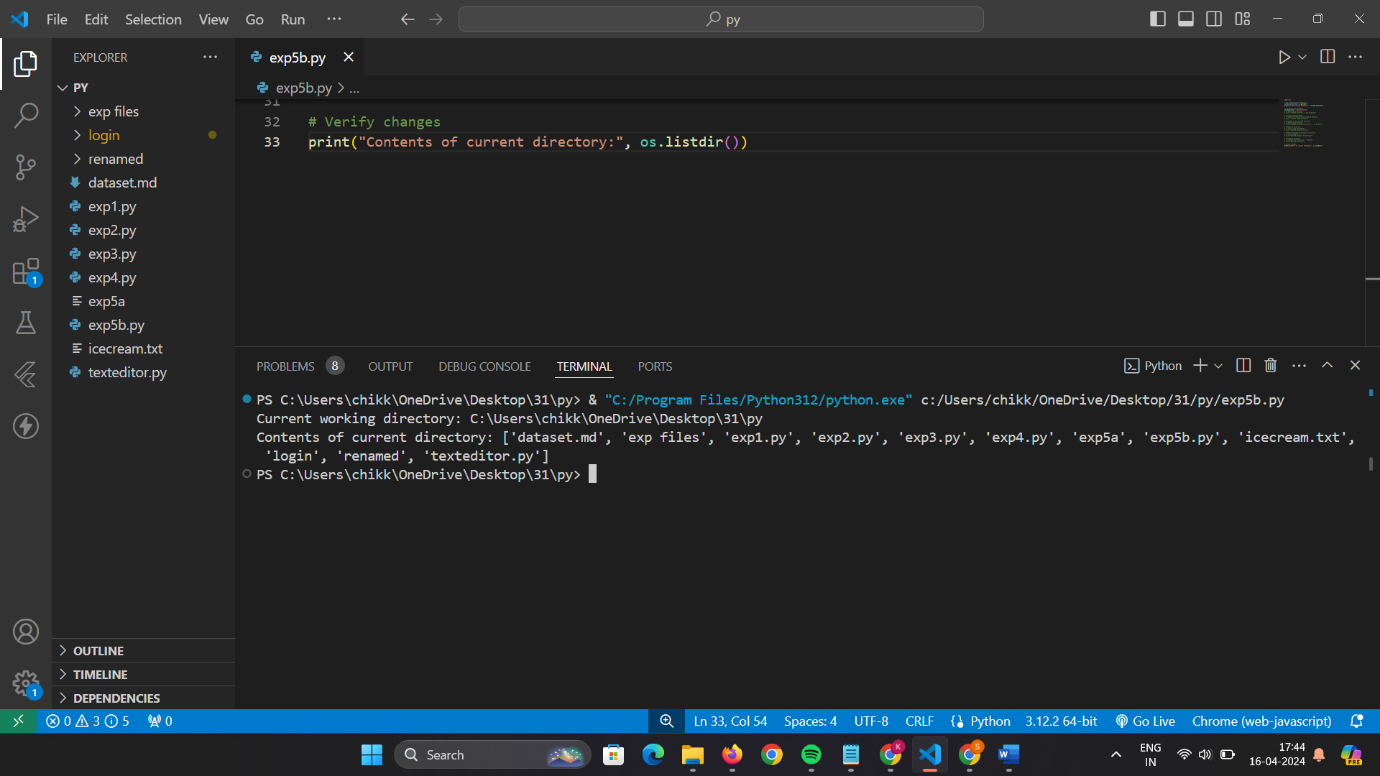
print("Contents of current directory:", os.listdir())

**Directories Output:**

****

****

****

****

**3] Regular Expression Program:**

import re

def main():

# Define a sample text

text = "Cinamoroll is the cutest sanrio character ever !! 31"

# Using search to find the first occurrence of one or more digits (\d+)

print("Using search with quantifier +:")

match = re.search(r'\d+', text)

if match:

print("First sequence of digits found:", match.group())

else:

print("No sequence of digits found")

# Using findall to find all occurrences of zero or more digits (\d\*)

print("\nUsing findall with quantifier \*:")

digits = re.findall(r'\d\*', text)

print("All sequences of digits found:", digits)

# Using match to match the pattern at the beginning of the string (Cinamoroll) with zero or more characters following it (.\*)

print("\nUsing match with quantifier \*:")

matched = re.match(r'Cinamoroll.\*', text)

if matched:

print("Match found at the beginning of the string with zero or more characters following it:", matched.group())

else:

print("No match found at the beginning")

# Using split to split the text based on zero or more spaces (\s\*)

print("\nUsing split with quantifier \*:")

words = re.split(r'\s\*', text)

print("Words:", words)

# Using sub to substitute sequences of digits (\d{2}) with 'X'

print("\nUsing sub with quantifier {m}:")

substituted\_text = re.sub(r'\d{2}', 'XOXO', text)

print("Substituted text:", substituted\_text)

# Using sub to substitute sequences of digits (\d{2,4}) with 'X'

print("\nUsing sub with quantifier {m,n}:")

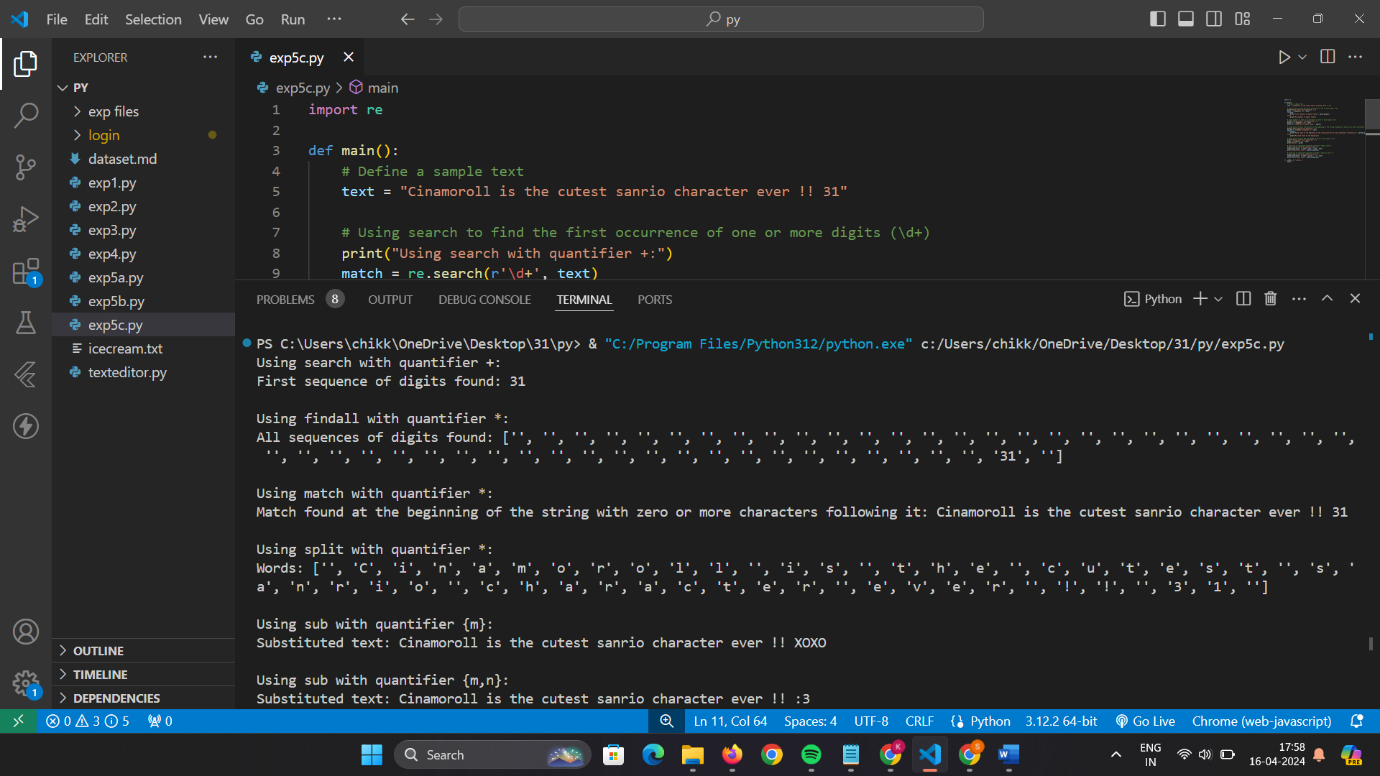
substituted\_text = re.sub(r'\d{2,4}', ':3', text)

print("Substituted text:", substituted\_text)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Regular Expression Output:**

****

**Conclusion:**

In this exploration of files, directories, and regular expressions in Python, we delved into essential aspects of file and directory handling, as well as text manipulation using regular expressions. We learned how Python facilitates operations such as creating, renaming, and removing directories, appending data to existing files, and reading from files. Additionally, we explored the powerful capabilities of regular expressions for pattern matching and text substitution within strings. This hands-on experience underscores Python's versatility and efficiency in handling various file-related tasks and text processing operations. Overall, this exploration provides valuable insights into Python's capabilities for effective file and directory management, as well as text manipulation, enhancing our understanding of practical programming applications.